UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/643,586	08/18/2003	Gregory J. Faanes	1376.699US1	4007

21186 7590 10/19/2006

SCHWEGMAN, LUNDBERG, WOESSNER & KLUTH, P.A.
P.O. BOX 2938
MINNEAPOLIS, MN 55402

EXAMINER	
FENNEMA, ROBERT E	

ART UNIT	PAPER NUMBER
2183	

DATE MAILED: 10/19/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	10/643,586	FAANES ET AL.	
	Examiner	Art Unit	
	Robert E. Fennema	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *01 August 2006*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-16* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-16* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *2/21/06;8/1/06*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-16 have been considered. Claims 12-16 have been added as per Applicant's request. Claims 1-3 and 5-11 have been amended as per Applicant's request.

Specification

2. In page 1 of the specification, in the Related Applications field, the Patent Application numbers have not been filled out. It is requested that the Applicant fills in the blanks as appropriate.

Claim Rejections - 35 USC § 112

3. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

4. Claims 1-16 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention. Examiner could not find support in the specification for the amended and new claimed subject matter in an examination of the specification, and no indication in the remarks was given on where to locate said claimed subject matter.

Claim Rejections - 35 USC § 102

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

5. Claim 1 is rejected under 35 U.S.C. 102(b) as being anticipated by Beard et al. (USPN 5,430,884, herein Beard).

6. As per Claim 1, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit (Column 3, Lines 1-7), wherein the vector processing unit includes a vector dispatch unit (Column 3 Lines 27-40), decoupling operation of the scalar processing unit from that of the vector processing unit, the method comprising:

sending a vector instruction from the scalar processing unit to the vector dispatch unit, wherein second includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

reading a scalar operand, wherein reading includes transferring the scalar operand from the scalar processing unit to the vector dispatch unit (Column 3, Lines 38-40 and Column 12, Lines 25-36);

predispatching the vector instruction within the vector dispatch unit if the vector instruction is scalar committed (Column 3, Lines 27-30);

dispatching the predispatched vector instruction if all required operands are ready (Column 3, Lines 30-34); and

executing the dispatched vector instruction as a function of the scalar operand (Column 12, Lines 26-40 disclose the scalar operands being put into a queue from the S registers. Generally in computing systems, the result of a scalar operation is not available outside the pipeline until it is committed, and there is no evidence in the specification to say otherwise, therefore the scalar operand in the queue would not be there until it is "scalar committed", which then allows the vector instruction to initiate, or begin execution).

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

8. Claims 2-4, 7, and 12-16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beard, in view of Patterson et al. (Herein Patterson).

9. As per Claim 2, Beard teaches: The method according to claim 1, wherein executing the dispatched vector instruction includes translating an address associated with the vector instruction (Column 31, Lines 3-8), but fails to teach:

and trapping on a translation fault.

While Beard does not explicitly teach trapping on a translation fault, Patterson teaches that a way to save the pipeline state safely in the event of an exception (also

known as a fault) is to insert a trap in the pipeline (Page 183). A translation fault is an example of an exception, and thus would require a trap in order for the processor to regain correct operating status. One of ordinary skill in the art would recognize the use of a trap in order to regain control of the system, which is required in order for the computer system to continue operating properly. Therefore, one of ordinary skill in the art at the time the invention was made would have recognized the need to trap in the event of a translation fault.

10. As per Claim 3, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, wherein the vector processing unit includes a vector dispatch unit,, a method of decoupling operation of the scalar processing unit from that of the vector processing unit, the method, comprising:

sending a vector instruction from the scalar processing unit to the vector dispatch unit, wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

reading a scalar operand, wherein reading includes transferring the scalar operand from the scalar processing unit to the vector dispatch unit (Column 11, Lines 44-53);

predispatching the vector instruction within the vector dispatch unit if the vector instruction is scalar committed (Column 3, Lines 27-30);

dispatching the predispatched vector instruction if all required operands are ready (Column 3, Lines 30-34);

generating an address for a vector load (Column 12, Lines 40-44 show how an address can be generated);

issuing a vector load request to memory (Column 23, Lines 24-26);

receiving vector data from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

executing the dispatched vector instruction on the vector data stored in the vector register (Column 3, Lines 27-40), but fails to teach:

storing the vector data in a load buffer;

transferring the vector data from the load buffer to a vector register.

While Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data

cache (which can function as a load buffer) into Beards invention to increase performance.

11. As per Claim 4, Beard teaches: The method according to claim 3, wherein the vector processing unit includes a vector execute unit (Column 3, Lines 1-7, both the scalar and vector processor has functional units, which execute instructions) and a vector load/store unit (Patterson, Page B-5, the vector load-store unit), wherein issuing a vector load request to memory includes issuing and executing vector memory references in the vector load/store unit when the vector load store unit has received the instruction and memory operands from the scalar processing unit (Column 3, Lines 27-40). While Beard did not explicitly disclose a vector load/store unit, Patterson discloses that it is an essential unit of a basic vector architecture to load and store to/from vectors. Therefore, a vector load/store unit would necessarily have to be present in Beards invention in order for it to function.

12. As per Claim 7, Beard teaches: A computer system, comprising:

a scalar processing unit (Column 3, Lines 1-7); and

a vector processing unit (Column 3, Lines 1-7) wherein the vector processing unit includes a vector dispatch unit, a vector execute unit and a vector load/store unit;

wherein the scalar processing unit sends a vector instruction and a scalar operand to the vector dispatch unit, wherein sending includes marking the vector

instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

wherein the vector dispatch unit predispatches the vector instruction within the vector dispatch unit if the vector instruction is scalar committed (Column 3, Lines 27-30) and then dispatches the predispatched vector instruction to one or more of the vector execute unit and the vector load/store unit if all required operands are ready (Column 3, Lines 30-34);

wherein the vector load/store unit receives an instruction and memory operands from the scalar processing unit, issues and executes a vector memory load reference as a function of the instruction and the memory operands receives from the scalar processing unit (Column 3 Lines 27-40), but fails to teach:

and stores data received as a result of the vector memory reference in a load buffer; and

wherein the vector execute unit issues the vector memory load instruction and transfers the data received as a result of the vector memory reference from the load buffer to a vector register.

While Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher

cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data cache (which can function as a load buffer) into Beards invention to increase performance.

13. As per Claim 12, Patterson teaches: The method according to claim 3, wherein storing the vector data in a load buffer and transferring the vector data from the load buffer to a vector register are decoupled from each other (Pages 375-381, the cache load is decoupled from a cache read).

14. As per Claim 13, Beard teaches: The method according to claim 3, but fails to teach: wherein storing the vector data in a load buffer includes writing memory load data to the load buffer until all previous memory operations complete without fault. However, Patterson teaches that out of order execution adds a large amount of complexity to a system, and that a large amount of hardware is required to handle it (Pages 241-243). Given the disadvantages of out-of-order execution, one of ordinary skill in the art would have been motivated to remove the out of order elements of the system, and make it a simpler in-order execution machine in order to reduce complexity and hardware cost,

which would require all previous memory operations to complete before the current one could begin.

15. As per Claim 14, Patterson teaches: The method according to claim 4, wherein storing the vector data in a load buffer and transferring the vector data from the load buffer to a vector register are decoupled from each other (Pages 375-381, the cache load is decoupled from a cache read).

16. As per Claim 15, Patterson teaches: The method according to claim 4, wherein storing the vector data in a load buffer includes writing memory load data to the load buffer until all previous memory operations complete without fault (Pages 241-243 teaches out-of-order execution and its disadvantages, providing a motivation to use in order execution instead, as described on Page 241).

17. As per Claim 16, Patterson teaches: The system according to claim 7, wherein the load buffer stores memory load data until it is determined that no previous memory operation will fail and, if no previous memory operations have failed, the load buffer transfers the data to the vector register (Pages 241-243 teaches out-of-order execution and its disadvantages, providing a motivation to use in order execution instead as described on Page 241).

18. Claims 5-6, and 8-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beard, in view of Gharachorloo et al (herein Gharachorloo).

19. As per Claim 5, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, wherein the vector processing unit includes a vector dispatch unit, a method of decoupling operation of the scalar processing unit from that of the vector processing unit, the method comprising:

sending a vector instruction from the scalar processing unit to the vector dispatch unit, wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

reading a scalar operand, wherein reading includes transferring the scalar operand from the scalar processing unit to the vector dispatch unit (Column 11, Lines 44-53);

predispatching the vector instruction within the vector dispatch unit if the vector information is scalar committed (Column 3, Lines 27-30);

dispatching the predispatched vector instruction if all required operands are ready (Column 3, Lines 30-34);

generating a first and a second address for a vector load (Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit, and Column 3 Lines 1-3 is one of many examples showing Beards invention is capable of

executing multiple instructions, and Column 12, Lines 26-28 show that a load can have multiple addresses);

issuing first and second vector load requests to memory (Column 23, Lines 24-26);

receiving vector data associated with the first and second addresses from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

storing vector data associated with the first address in a first vector register (Column 23, Lines 24-33);

storing vector data associated with the second address in a second vector register (Column 23, Lines 24-33);

executing a vector instruction on the vector data stored in the first vector register (Column 3, Lines 27-40);

executing the dispatched vector instruction on the vector data stored in the second vector register (Column 3, Lines 27-40), but fails to teach:

renaming the second vector register.

While Beard does not explicitly teach renaming, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic

scheduling of instructions. The reorder buffer that comes with that eliminates storage conflicts through register renaming (Page 5, Section 4.2). The reorder buffer also allows for storage of speculative results, to potentially increase the workload of the processor. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer (and thus being able to rename registers) to implement the out-of-order execution in Beards invention to increase performance.

20. As per Claim 6, Beard teaches: The method according to claim 5, wherein the vector processing unit includes a vector execute unit (Column 3, Lines 1-7, both the scalar and vector processor has functional units, which execute instructions) and a vector load/store unit (Patterson, Page B-5, the vector load-store unit), wherein issuing a vector load request to memory includes issuing and executing vector memory references in the vector load/store unit when the vector load store unit has received the instruction and memory operands from the scalar processing unit (Column 3, Lines 27-40). While Beard did not explicitly disclose a vector load/store unit, Patterson discloses that it is an essential unit of a basic vector architecture to load and store to/from vectors. Therefore, a vector load/store unit would necessarily have to be present in Beards invention in order for it to function.

21. As per Claim 8, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of decoupling scalar and vector execution, comprising:

sending a vector instruction that requires scalar operands to the scalar instruction queue and to a vector instruction queue (The scalar required part of the instruction goes into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36), wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes generating an address and writing the address to a scalar operand queue (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit);

predispatching the vector instruction sent to the vector instruction queue if the vector instruction is scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the address is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the dispatched vector instruction in the vector processing unit, wherein executing the vector instruction in the vector processing unit includes reading the address from the scalar operand queue and generating a memory request as a function of the address read from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit), but fails to teach:

dispatching scalar instructions to a scalar instruction queue.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have

used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

22. As per Claim 9, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of decoupling scalar and vector execution, comprising:

sending a vector instruction that requires scalar operands to the scalar instruction queue and to a vector instruction queue (The scalar required part of the instruction goes into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36), wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes writing a scalar operand to a scalar operand queue (Column 3, Lines 38-40);

predispatching the vector instruction sent to the vector instruction queue if the vector instruction is scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the scalar operand is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction if all required scalar operands are ready (Column 3, Lines 30-34); and

executing the dispatched vector instruction in the vector processing unit, wherein executing the vector instruction in the vector processing unit includes reading the scalar operand from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution), but fails to teach:

dispatching scalar instructions to a scalar instruction queue.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

23. Claims 10 and 11 are rejected under 35 U.S.C. 103(a) as being unpatentable over Beard and Gharachorloo, further in view of Patterson.

24. As per Claim 10, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of executing a vector instruction, comprising:

sending a vector instruction to the scalar instruction queue and to vector instruction queue (The scalar required part of the instruction goes into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36) wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

executing the vector instruction in the scalar processing unit, wherein executing the vector instruction in the scalar processing unit includes generating an address and writing the address to a scalar operand queue (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit);

predispatching the vector instruction sent to the vector instruction queue if the vector instruction is scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the address is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar register

scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction if all required operands are ready (Column 3, Lines 30-34); and

executing the dispatched vector instruction in the vector processing unit (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution), wherein executing the dispatched vector instruction in the vector processing unit includes:

reading the address from the scalar operand queue (Column 12, Lines 33-36 show the vector register getting the operand (in this case the address) as it begins);

generating a memory request as a function of the address read from the scalar operand queue (Column 3 Lines 27-40, wherein all memory requests are a function of the address, and all addresses must necessarily come from the scalar unit, and thus the scalar operand queue for the vector to use it. Also see Column 23, Lines 24-26);

receiving vector data from memory (Column 23, Lines 24-26, when a load is requested from memory, it has to be received);

executing a vector instruction on the vector data storing in the vector register (Column 3, Lines 27-40), but fails to teach:

dispatching scalar instructions to a scalar instruction queue;

storing the vector data in a load buffer;

transferring the vector data from the load buffer to a vector register.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

Furthermore, while Beard teaches an instruction cache, which consists of multiple buffers (Column 7, Lines 52-55), he does not explicitly teach a data cache, which is commonly used in computer systems to vastly increase performance in loads and stores. Patterson teaches that most computers have 2 levels of caches now, both instruction and data caches, with Pages 380-381 showing an example data cache. As it is well known in the art, the cache would read the data directly from memory (or from a higher cache level), and the computer system would then read the data from the cache (buffer) into the appropriate register(s). One of ordinary skill in the art would have recognized the need to add a data cache into Beards system, for the same reason there

is an instruction cache, increased performance and faster memory reads. Therefore, one of ordinary skill in the art at the time the invention was made would have added a data cache (which can function as a load buffer) into Beards invention to increase performance.

25. As per Claim 11, Beard teaches: In a computer system having a scalar processing unit and a vector processing unit, a method of unrolling a loop, comprising:

preparing a first and a second vector instruction (Column 3 Lines 1-3 is one of many examples showing Beards invention is capable of executing multiple instructions),

sending the first and second vector instructions to the scalar instruction queue and to a vector instruction queue (The scalar required part of the instruction does into the scalar queue, and the vector queue is disclosed in Column 3, Lines 34-36), wherein sending includes marking the vector instruction as complete if the vector instruction is not a vector memory instruction and if the vector instruction does not require scalar operands (Column 3, Lines 27-30);

executing each vector instruction in the scalar processing unit (Column 3, Lines 38-40. In addition, Column 12, Lines 40-44 show an example of an address being generated in the scalar processing unit),

predispatching each vector instruction sent to the vector instruction queue if the vector instruction is scalar committed (Column 3, Lines 27-30);

notifying the vector processing unit that the scalar operand is available in the scalar operand queue (Column 11, Lines 44-53. It is notified by a bit in the scalar

register scoreboard. When the vector processing unit looks into the scoreboard, it can determine if the scalar operand is available);

dispatching the predispatched vector instruction if all required operands are ready (Column 3, Lines 30-34); and

executing the first and second dispatched vector instructions in the vector processing unit, wherein executing the dispatched vector instructions in the vector processing unit includes reading the scalar operands associated with each instruction from the scalar operand queue (Column 12, Lines 26-36, the scalar operand data is transferred to the vector register unit as it begins execution, and is done only when each instruction needs it), but fails to teach:

wherein each vector instruction execute an iteration through the loop and wherein each vector instruction requires calculation of a scalar loop value;

wherein executing each vector instruction in the scalar processing unit includes writing a scalar operand representing the scalar loop value calculated for each vector instruction to a scalar operand queue.

While Beard does not explicitly teach a queue to put scalar instructions in before execution, Gharachorloo teaches of a reservation station and a reorder buffer (Page 5, Section 4.2). A reservation station is well known in the art as a way for a processor to execute instructions out of order, and Beard does teach an out-of-order machine (Column 23, Lines 20-21). A reservation station acts as a queue or buffer to hold decoded instructions before execution. It allows the instruction decoding to be decoupled from instruction execution, allowing dynamic scheduling of instructions. One

of ordinary skill in the art would recognize the advantage of dynamic scheduling of instructions as potentially bypassing unnecessary conflicts (along with the use of a reorder buffer) and increasing performance, and to also allow speculative execution. Therefore, one of ordinary skill in the art at the time the invention was made would have used a reservation station and a reorder buffer to implement the out-of-order execution in Beards invention to increase performance.

While Beard does not also explicitly teach a vector instruction, each of which each vector instruction executes an iteration of a loop, Patterson gives an example of how vector instructions can be used for this purpose (Pages B-7 and B-8). In this example, a MULTSV instruction is used for the first part, and an ADDV instruction for the second, although one of ordinary skill in the art could also see how this would be able to be modified to be done in one or multiple vector instructions, depending on the required steps of the loop in question. It can also be seen that in this example, "a" is a scalar operand required for the vector instructions to execute. One of ordinary skill in the art would be able to see the value in using vector instructions to execute this loop, as shown by the vastly decreased code size as shown in the example on B-8. Therefore, one of ordinary skill in the art at the time the invention was made would have recognized the advantage of using vector instructions for the purpose of executing loops in Beards invention, to decrease code size, and therefore increase performance.

Response to Arguments

26. Applicant's arguments filed 8/1/2006 have been fully considered but they are not persuasive.

As per Claim 1, Applicant has argued that Beard does not teach the two-step approach for dispatching vector instructions as described by the Applicant. However, Beard does teach that dispatching a vector instruction is a two-part method, in that it must be both issued and initiated before execution, as seen in Column 3, Lines 21-30. The instruction is issued when scalar data is available (or if it does not require any), and is then initiated once the resources for its execution are available (Column 3, Lines 60-66). Thus, there are two steps to a vector being executed, first, collecting the operands, and second, securing the resources.

Regarding Claims 2-4, 7, and 12-16, the same arguments apply, as Beard does teach the two-step method.

Regarding Claims 5-6 and 8-11, Examiner refers to the above arguments and the previous and current office action for why Beard and the combinations of references used in addition to Beard teach the claimed subject matter in question.

Conclusion

27. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Robert E. Fennema whose telephone number is (571) 272-2748. The examiner can normally be reached on Monday-Friday, 8:00-4:30.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Robert E Fennema
Examiner
Art Unit 2183

RF

EDDIE CHAN
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100